ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Ingeniería del Software

Sistema web para la recopilación y análisis de datos sobre Twitter

A Web-based System for Data Collection and Analysis over Twitter

Realizado por

Sergio Cuenca Avilés

Tutorizado por

Eduardo Guzmán de los Riscos

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

MÁLAGA, Junio 2017

Fecha defensa:

El Secretario del Tribunal

Resumen: En este proyecto se expone una herramienta para la extracción de tuits y el cálculo de estadísticas. Para la extracción de los tuits se ha utilizado la API Twitter4j y para el almacenamiento de dichos tuits MongoDB. El dibujo de las gráficas se ha realizado con la librería de JavaScript C3.js. La herramienta posee funcionalidades que requiere de una tarea periódica y de otra tarea que se ejecuta una vez llegado una fecha determinada. Para satisfacer estas dos tareas se ha utilizado el Servicio EJB Timer Service. Las vistas han sido realizadas mediante la combinación Bootstrap y de Java Server Faces. Los datos que son aportados por la aplicación son los tres hashtags más usados, las tres cuentas más mencionadas, la hora a la que se ha publicado el tuit y una valoración sobre un tema determinado. Para obtener la valoración se realiza una petición POST a un Servicio REST al que se le manda un texto y en la respuesta aparece entre otros datos si la valoración es negativa o positiva.

Palabras claves: Recuperación de la Información, Twitter, Twitter4j, MongoDB, C3.js, EJB Timer Service

Abstract: This project presents a tool for the extraction of tweets and the calculation of statistics. Twitter4j API has been used for the extraction of tweets and MongoDB for the storage of it. The drawing of the graphs was done with C3.js JavaScript library. The tool has functionalities that require a periodic task and another task that is executed at certain date. EJB Timer Service has been used to implement these two tasks. The views have been developed using Bootstrap and Java Server Faces technologies. The data computed by the application are the three most used hashtags, the three most mentioned accounts, the time at which the tweet was published and a valuation on a particular topic. To obtain the valuation, a POST request is made to a REST Service with text of the tweet and as a response, among other information, the opinion trend is shown.

Keywords: Information Retrieval, Twitter, Twitter4j, MongoDB, C3.js, EJB Timer Service.

Índice

1. Introducción

Vivimos en una sociedad cambiante y en continua comunicación. En este contexto, surge la necesidad de desarrollar herramientas que nos permitan analizar las comunicaciones que se producen y extraer, a partir de ellas, información relevante. Para realizar este trabajo de forma eficaz, contamos con un aspecto a favor y en creciente desarrollo: las redes sociales. Tomando todo esto en conjunto y trabajando en la red social de Twitter, nos planteamos desarrollar una aplicación web capaz de monitorizar y extraer datos relevantes a partir de tuits en relación a un tema determinado, así como de realizar análisis y aportes de estadísticas. Estos análisis pueden ser de especial relevancia en el ámbito empresarial especialmente para el desarrollo de campañas de marketing dirigidas en base a la opinión de los usuarios sobre un determinado tema o producto.

1.1 Objetivos

El objetivo de este proyecto es desarrollar una herramienta web que permita la extracción y análisis de información sobre la red social Twitter. Para ello, dicha herramienta almacenará tuits con aspectos en común en relación a un mismo tema, permitiendo al usuario indicar el periodo de tiempo que desee monitorizar. Una vez extraída la información que el usuario necesita, se aportarán una serie de estadísticas. El ámbito de aplicación del proyecto será el marketing, de forma que se intentarán obtener los gustos de los usuarios que siguen a la cuenta de una determinada empresa o producto, canales más adecuados de publicidad en función de las preferencias de aquellos usuarios que muestran discrepancias o descontento con la marca o el producto, relaciones no apreciables a simple vista entre marcas y productos, etc. En resumen, se intentará dotar a la herramienta de funcionalidades que permitan estudiar perfiles poblacionales (segmentos de población, localización geográfica, etc.) y preferencias de los seguidores de un producto. La API de acceso a Twitter presenta diversas limitaciones con las que se deberá lidiar a lo largo del proyecto para conseguir los objetivos establecidos.

1.2 Contenido de la memoria en capítulos

Esta memoria está organizada de una forma similar a las etapas que sigue cualquier desarrollo software.

Tecnologías utilizadas: En este capítulo se van a exponer las tecnologías utilizadas en el desarrollo, así como una breve explicación de cada una de ellas. Entre las tecnologías explicadas se encuentra MongoDB, Bootstrap, Java Server Faces (JSF), etc.

Metodología utilizada: Durante esta fase se explicará la metodología usada para el desarrollo del proyecto.

Análisis de requisitos: En esta parte se procederá a exponer los requisitos que debe tener nuestra aplicación, así como los diagramas utilizados para definir el sistema.

Implementación: En el capítulo de la implementación se explicarán todos los elementos interesantes y más problemáticos de todo el código del proyecto.

Pruebas: Aquí se expondrán las diferentes pruebas que se han realizado y la solución a los problemas que han surgido de esas pruebas

Conclusiones: En este último capítulo aparecen las conclusiones obtenidas gracias al desarrollo del presente proyecto.

1.3 Metodología

La metodología empleada para el desarrollo del proyecto será incremental iterativa. Dadas las características de este proyecto que lo hacen cercano al ámbito de la investigación, esta metodología permitirá general diversos prototipos, evaluar sus resultados y rendimiento y, en base a la información obtenida, añadir nuevas funcionalidades en sucesivas versiones.

1.4 Tecnologías empleadas

En este punto aparecen las tecnologías usadas en el proyecto.

API Twitter4j: Librería de Java no oficial que permite integrar funcionalidades relacionadas con esta red social, compatible desde Java 5 y también es compatible con Android y Google App Engine.

API Streaming: Librería de Java no oficial que permite integrar funcionalidades para monitorizar esta red social.

NetBeans: Entorno de desarrollo

Java EE: Plataforma de programación para desarrollar aplicaciones empresariales distribuidas escritas en Java y que se ejecutan en un servidor de aplicaciones (glassfish en nuestro caso).

JavaServer Faces (JSF): *Framework* para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario.

MongoDB: Base de datos no relacional orientada a documentos, estos documentos son almacenados en el formato BSON (Representación binaria de JSON).

Homebrew: Gestor de paquetes para MacOS que permite instalar paquetes o complementos que no vienen de serie.

Sentiment Analysis: Servicio REST que manda en forma de respuesta un texto en formato JSON si un texto es positivo o negativo. En un primer momento se pensó utilizar API.AI pero su uso más complejo desechó esta opción.

Bootstrap: *Framework* para el desarrollo de interfaces gráficas de forma sencilla e intuitiva. Permite combinar muchos elementos de CSS, HTML5 o JavaScript. Está creado por Twitter y es de código abierto.

C3: Librería JavaScript basada en D3, otra librería JavaScript, para la realización de gráficas de forma sencilla.

1.5 Instalación

Para la utilización de la herramienta es necesario instalar MongoDB, en este caso se realizará la instalación en MacOS. Una vez que tenemos instalado *Homebrew* solo es necesario poner el siguiente comando en la consola:

```
brew install mongodb
```

Una vez realizado ya tendríamos MongoDB en nuestro ordenador.

Otro servicio a tener instalado es Glassfishh (Suele ser instalado en conjunto con NetBeans).

1.6 Herramientas existentes

En la siguiente tabla podemos ver una relación de herramientas ya existentes que pueden tener relación que nuestro proyecto.

Nombre	**Principales funcionalidades**
Audiense	Gratuita para cuentas con menos de 5000 seguidores. Esta herramienta es capaz de aportar estadísticas sobre las palabras más usadas en tu comunidad, así como estadística acerca de tus seguidores. Permite realizar búsquedas basadas en geolocalización.
Manage Flitter	Versión gratuita sin límites de seguidores. Permite conocer cuántas cuentas de entre aquellas que te siguen están inactivas y dejar de seguirlas si queremos. También te da la posibilidad de seguir a las cuentas que te han mencionado en algún momento. Aporta

	estadísticas acerca del día y la hora en la que tus tuits son más vistos.
CrowdFire	Como la anterior, permite dejar de seguir a aquellas cuentas que son inactivas. También te ofrece la opción mandar mensajes directos de forma automática a través de ella y aporta estadísticas acerca de cuentas de *influencers* para que puedas seguirlas
Twitter Analytics	Esta herramienta aporta información sobre tus tuits como por ejemplo el número de tuits que has publicado, el número de visitas a tu perfil, qué publicación ha tenido más interacciones o la cuenta que te sigue que a su vez tiene más seguidores
TweetBinder	Esta herramienta permite ver la analítica de un hashtag o una cuenta en los últimos 300 tuits.
Tweriod	Ofrece estadísticas muy simples y precisas acerca del impacto que tiene publicar a una hora u otra.

Tabla 1: Tecnologías existentes

1.7 Limitaciones

Durante la realización del presente proyecto aparecen varias limitaciones que se exponen a continuación.

Twitter4j sólo nos permite obtener tuits con menos de 7 días de antigüedad, además, también establece una limitación en cuanto a la cantidad de tuits que podemos extraer en un periodo temporal. En concreto, y de forma aproximada, nos restringe a unos 3500 tuits cada 15 minutos.

En cuanto al *streaming* nos encontramos con una limitación variable, Twitter nos permite obtener hasta un 1% de los tuits publicados en ese momento por segundo. Por tanto, debemos ajustar las peticiones de tuits para que Twitter nos corte el *streaming* la menor cantidad posible de veces, manteniendo además un equilibrio entre los temas con mucha afluencia de tuits y los temas de menor cantidad.

Para el procesamiento del lenguaje natural del tuit utilizamos un servicio REST, que nos limita a enviar los tuits en inglés. Por otra parte, aunque no es conocido el número límite de peticiones diarias, es necesario establecer un límite, lo cual hace imposible el procesado de todos los tuis en la elaboración de esta estadística.

2. Análisis de Requisitos

Antes de exponer la lista de requisitos es necesario explicar un poco el contexto en el que se mueve la aplicación.

Twitter es, actualmente, una de las herramientas sociales más utilizadas en multitud de ámbitos. Oferta de servicios, promoción de productos… son solo algunos ejemplos de usabilidad que esta red social nos ofrece. Es por esto por lo que se plantea la posibilidad de desarrollar una herramienta que permita ofrecer un análisis completo de distintos aspectos como puede ser la valoración de los usuarios, las horas en las que se han publicado el mayor número de tuits o las cuentas y hashtags más usados en relación a un tema. Parece adecuado también la introducción en la herramienta la posibilidad de que el usuario elija el periodo temporal en el que realizar el análisis.

2.1 Requisitos funcionales

ID	Nombre	Descripción
TW1	Introducir un tema	El usuario podrá introducir un tema
TW2	Introducir una fecha de finalización	El usuario podrá introducir una fecha de finalización
TW3	Iniciar el *streaming*	El usuario podrá iniciar el *streaming*
TW4	Parar el *streaming*	El usuario podrá parar el *streaming*
TW5	Visualizar Datos	El usuario podrá visualizar los datos

TW6	Introducir un tema para analizar tuits pasados	El usuario podrá introducir un tema para realizar un análisis de tuits publicados en el pasado

Tabla 2: requisitos funcionales

2.2 Requisitos no funcionales

ID	Nombre	Descripción
TWN1	Calcular Datos	El sistema deberá realizar un cálculo de los datos de forma periódica
TWN2	Parar el *Streaming* automáticamente	El sistema deberá parar el *Streaming* a una hora determinada
TWN3	Almacenar tuits	El sistema deberá almacenar los tuits en una base de datos
TWN4	Navegador	El sistema deberá funcionar en los navegadores Chrome, Firefox, Safari e Internet Explorer
TWN5	Múltiple recogida de datos	El sistema debe ser capaz de realizar una obtención de datos a través de tuits pasados, a la misma vez que está realizando una monitorización.

Tabla 3: Requisitos no funcionales

3. Diseño

En este apartado se van a exponer los puntos relacionados con el diseño de la aplicación.

3.1 Actores

Los actores que aparecen en el ámbito de nuestra aplicación son los siguientes expuestos:

- **Usuario:** Cualquier persona que tenga acceso a la aplicación.
- **Sistema:** Representa la máquina donde se ejecuta el planificador.

3.2 Diagramas de casos de usos

En este apartado se representarán los diagramas de Casos de Uso.

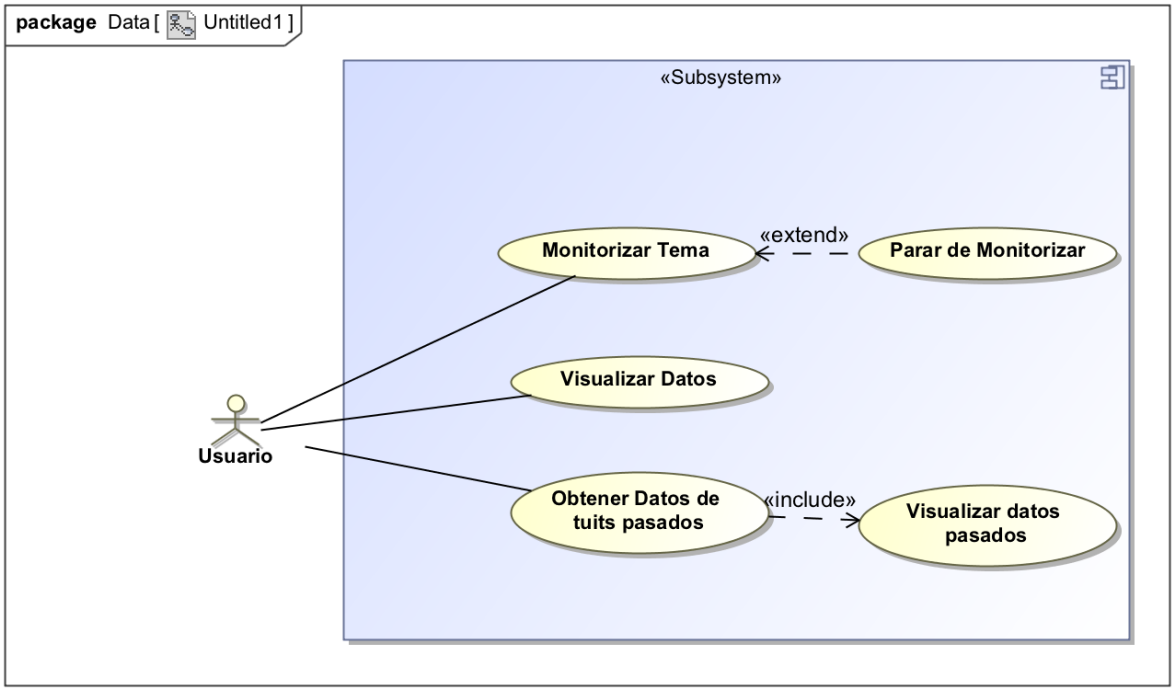

Figura 1: Diagrama de caso de uso 1

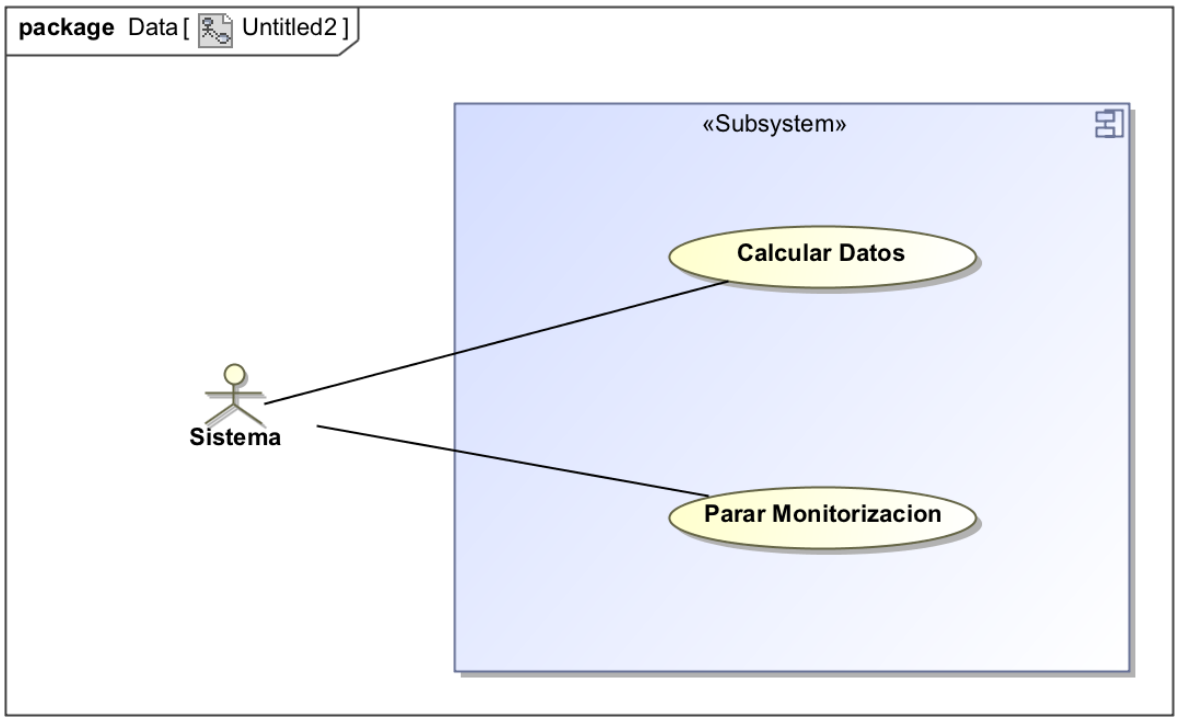

Figura 2: Diagrama de caso de uso 2

A continuación vamos a expandir los escenarios principales más interesantes:

Caso de uso: Monitorizar Tema

Actores: Usuario

Descripción: Activar el *streaming* de un tema determinado

Precondiciones: -

Escenario Principal:

1. El usuario accede a la aplicación
2. El usuario introduce el tema deseado
3. El usuario introduce la fecha de finalización deseada
4. El usuario pulsa el botón monitorizar
5. El sistema comienza a monitorizar

Escenario Alternativo 1:

2. El usuario introduce la fecha de finalización deseada

3. El usuario introduce el tema deseado

Escenario Alternativo 2:

6. El usuario pulsa el botón Parar

Escenario de Error:

6. El sistema muestra un mensaje de error diciendo que la fecha o el tema no son válidos.

Caso de uso: Visualizar Datos

Actores: Usuario

Descripción: El usuario podrá visualizar los datos calculados

Precondiciones: -

Escenario Principal:

1. El usuario pulsa el botón Datos
2. El sistema muestra los datos calculados

Escenario Alternativo 1:

2. El sistema muestra un aviso de que no hay datos calculados

Caso de uso: Monitorizar Tema Pasado

Actores: Usuario

Descripción: Obtener datos de tuits pasados

Precondiciones: -

Escenario Principal:

1. El usuario accede a la aplicación
2. El usuario pulsa el botón de datos pasados
3. El usuario introduce el tema deseado
4. El usuario introduce la fecha de finalización deseada
5. El usuario pulsa el botón monitorizar
6. El sistema comienza a monitorizar
7. El sistema muestra los datos obtenidos

Escenario Alternativo 1:

2. El usuario introduce la fecha de finalización deseada
3. El usuario introduce el tema deseado

Escenario de Error:

7. El sistema muestra un mensaje de error diciendo que el tema no es válido

3.3 Modelo de Clases

En este apartado se representa el diagrama de Clases del sistema con los atributos más significativos:

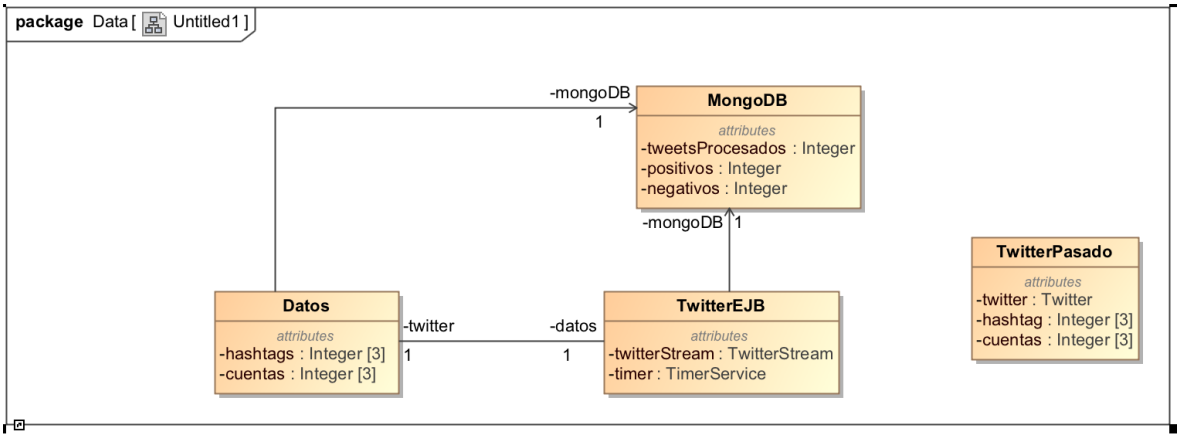

Figura 4: Diagrama de clases

Como vemos en la figura anterior, nuestro modelo es bastante sencillo. Se va a proceder a explicar la parte más importante del funcionamiento de cada clase.

- **MongoDB:** Clase encargada de interactuar con la base de datos. introducirá un tuit en la base de datos

- **TwitterEjB:** Esta clase implementa el *streaming* con todas sus funcionalidades

- **Datos:** Clase que almacena los datos.

- **TwitterPasado:** En esta clase se realizará todo el proceso para procesar obtener los tuits que fueron publicado con anterioridad

3.4 Diagrama de Secuencia

En este apartado podemos ver un diagrama de secuencia para entender mejor cómo funciona el sistema a la hora de realizar una monitorización y la visualización de datos:

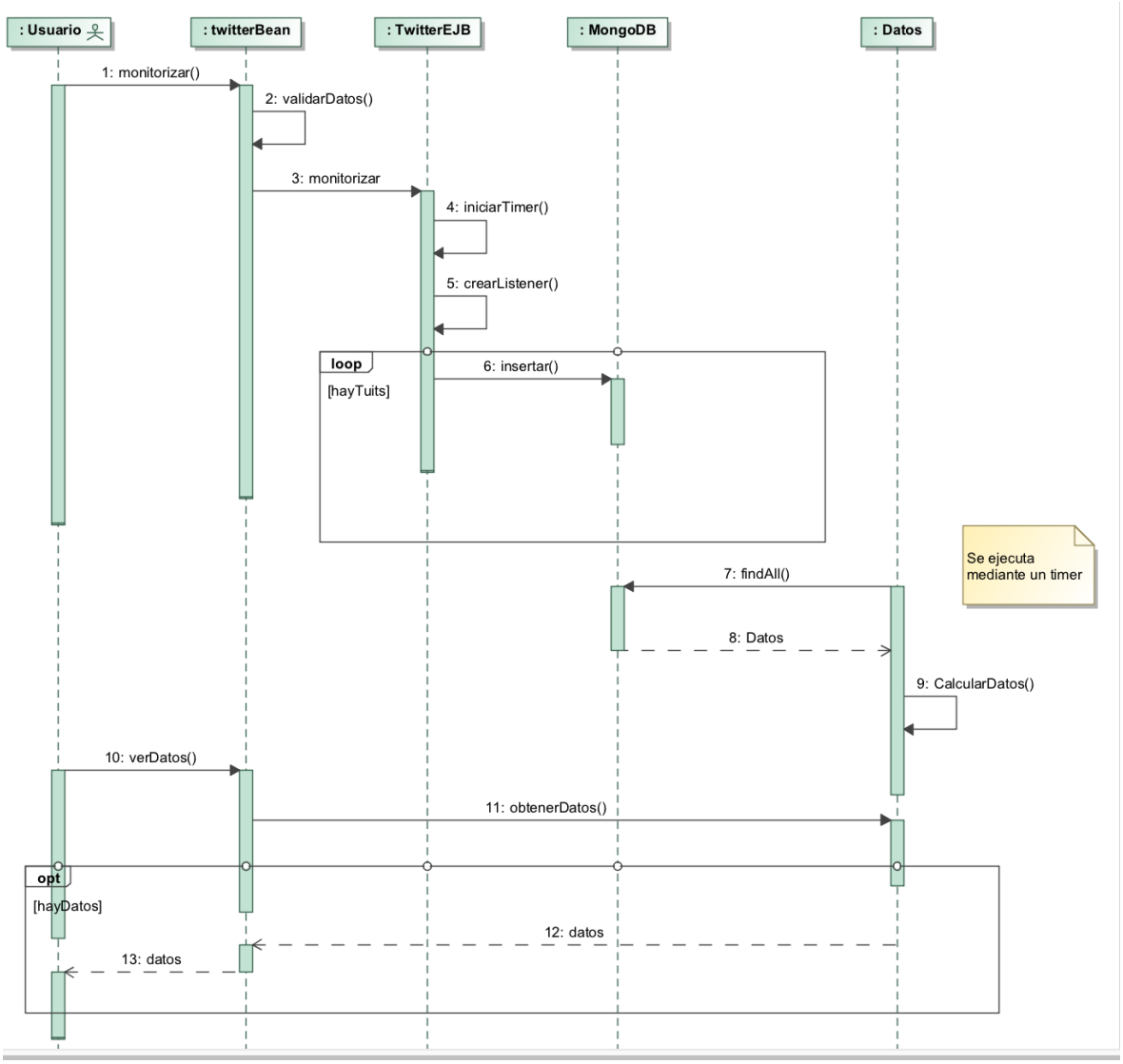

Figura 5: Diagrama de secuencia

A continuación, se va explicar la secuencia que el diagrama anterior representa.

Una vez que el usuario ha introducido los datos para monitorizar y el sistema ha validado dichos datos, el sistema creará el *listener* e iniciará el *timer* para establecer la fecha de parada. Los tuits llegarán mediante el *listener* uno a uno y estos serán introducidos en la base de datos (representado como un bucle en el diagrama anterior). Más tarde, se ejecutará un *timer* (distinto al de parada) que calculará los datos obtenidos hasta el momento. Para ello, buscará en la base de datos todos los datos almacenados y procesará los datos que no hayan sido procesados con anterioridad. Por último, el usuario solicitará ver los datos y estos serán mostrados por el sistema.

4. Implementación

En este apartado se expondrán los puntos problemáticos y aquellos que han resultado ser más interesantes del desarrollo del proyecto.

4.1 Streaming

Para la realización del *Streaming* se ha utilizado la API Twitter4j, la cual ofrece entre otras cosas la posibilidad de monitorizar un tema. Todas las acciones a realizar se tratan a través de un objeto del tipo *TwitterStream*.

Para usar esta API es necesario autenticarnos mediante OAuth, lo cual implica tener creada una cuenta de Twitter, así como un proyecto de desarrollo asociado a la cuenta para poder obtener nuestra *API KEY* y nuestro *TOKEN*.

```
ConfigurationBuilder cb2 = new ConfigurationBuilder();
cb2.setDebugEnabled(true)
        .setOAuthConsumerKey("XXXXX")
        .setOAuthConsumerSecret("XXXX")
        .setOAuthAccessToken("XXXXX")
        .setOAuthAccessTokenSecret("XXXXX")
        .setHttpProxyPort(3128)
        .setHttpProxyHost("proxy.wifi.uma.es")
        .setHttpProxyUser("proxy.wifi.uma.es");
this.twitterStream = new TwitterStreamFactory(cb2.build()).getInstance();
```

Figura 6: código de Streaming

En caso de ser necesario podemos configurar los datos para la utilización de un Proxy.

Una vez que se realiza correctamente la autenticación, se debe crear un *listener* que será el encargado de recibir los tuits sobre un tema determinado. Para crear el *listener* es necesario implementar los siguientes métodos:

- *onStatus(Status status)*: Método encargado de recibir los tuits (Status) y de realizar la lógica deseada con él. En nuestro caso almacenaremos la información necesaria en MongoDB.

- *onDeletionNotice(StatusDeletionNotice sdn)*: Método que nos permite realizar cualquier acción en caso de que algún tuit que contenga el tema deseado sea borrado.

- *onTrackLimitationNotice(int i)*: Este método devuelve un mensaje cada vez que un flujo limitado se vuelve ilimitado. Si este número es alto significa que se debe realizar una búsqueda más específica.

- *onScrubGeo(long l, long l1)*: Devuelve mensajes de eliminación de ubicación.

- *onStallWarning(StallWarning sw)*:Devuelve mensajes de advertencia.

- *onException(Exception excptn)*: Devuelve las excepciones que se hayan podido producir en el *Streaming.*

Para activarlo se debe crear un filtro al que le introducimos el tema en cuestión y se le añade al objeto *TwitterStream* junto con el *listener.*

```
this.listener = new StatusListener() {
    @Override
    public void onStatus(Status status) {
        try {
            almacenarTweet(status);
        } catch (InterruptedException ex) {
            System.err.println(ex.toString());
        }
    }

    @Override
    public void onDeletionNotice(StatusDeletionNotice sdn) {
    }

    @Override
    public void onTrackLimitationNotice(int i) {
        System.out.println("Limite " + i);
    }

    @Override
    public void onScrubGeo(long userId, long upToStatusId) {
    }

    @Override
    public void onStallWarning(StallWarning sw) {
        System.err.println("sw");
    }

    @Override
    public void onException(Exception excptn) {
        System.err.println(excptn.toString());
    }
};
FilterQuery filtro = new FilterQuery();
filtro.track(tema);

this.twitterStream.addListener(listener);
this.twitterStream.filter(filtro);
```

Figura 7: Código del Listener

4.2 Uso de la Base de Datos

La base de datos elegida en este proyecto ha sido MongoDB. Esta decisión ha sido tomada entre otros motivos por su simpleza tanto en la instalación y del uso como en la estructura a almacenar.

MongoDB trabaja a través de documentos que son almacenados en BSON, esto es, la representación binaria de JSON. Los documentos se agrupan en una

colección (concepto similar a tabla en las bases de datos relacionales) y cada documento puede tener un esquema distinto. En nuestro caso, utilizaremos una colección llamada “tweets” en la que guardaremos documentos con el texto, el id, la fecha y el usuario creador del tuit.

Para insertar un tuit primero extraemos los datos deseados del objeto y luego lo introducimos mediante el método *put* que funciona con un par clave (nombre del dato a almacenar) valor (propio dato).

```
DBObject documento = new BasicDBObject();
Date fechaCreacion = tweet.getCreatedAt();
DateFormat format = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
String fecha = format.format(fechaCreacion);
documento.put("fecha", fecha);
documento.put("id", tweet.getId());
documento.put("tweet", tweet.getText());
documento.put("usuario",tweet.getUser().getScreenName());
database.getCollection("tweets").insert(documento);
```

Figura 8: Código de base de datos

Ya que uno de los requisitos del sistema consiste en la realización de un cálculo periódico de los datos hasta el momento, es necesario extraer también de forma periódica los documentos almacenados en la base de datos. La extracción se realizará mediante un cursor que irá iterando por toda la colección y extrayendo uno por uno cada documento. Para evitar que se procesen tuits ya procesados se llevará un contador con el número de los que han sido procesados.

```
DBCollection col = database.getCollection("tweets");
DBCursor cur = col.find();

while(cur.hasNext()){
    DBObject document = cur.next();

salida = new JSONObject(JSON.serialize(document));
tweet = (String) salida.get("tweet");
fecha = (String) salida.get("fecha");
```

Figura 9: Código del find de la bbbd

4.3 Tareas con temporizador

Como ya se ha mencionado anteriormente, la aplicación tendrá varias tareas que deberán ejecutarse tras un periodo de tiempo y para ello haremos uso de los servicios *Timers* de Enterprise JavaBeans (EJBs).

La primera tarea que se va a tratar será la de parar el *streaming* a la hora que ha indicado el usuario anteriormente. Esta tarea no es periódica ya que se ejecutará una vez en cada monitorización. Para ello necesitaremos un objeto del tipo *TimerServices*, un método que inicie el *timer*, un método marcado con la anotación *@Timeout* y un método que elimine los temporizadores que se vayan utilizando una vez finalizada la tarea para ahorrar memoria. El objeto *TimerService* se declara mediante la etiqueta *@Resource*.

```
@Resource
TimerService timer;
```

Figura 10: Código del temporizador 1

Para iniciar el *timer* debemos crear un *TimerConfig*, en el que se pueden modificar la configuración pero que en nuestro caso mantendremos por defecto, y crear una acción simple mandando como parámetros la fecha deseada y la configuración antes mencionada.

```
TimerConfig timeconf = new TimerConfig();
timer.createSingleActionTimer(fecha, timeconf);
```

Figura 11: Código del temporizador 2

Una vez llegada la fecha marcada se ejecutará el método etiquetado como *@Timeout* en el que entre otras cosas se eliminará el temporizador utilizado y se detendrá el *Streaming.* Esto se realizará utilizando el objeto anteriormente creado (TwitterStream*),* borrando el *listener* y deteniendo el servicio. A la hora de parar el servicio puede darse una situación en la que queden tuits en cola para almacenar y se debe decidir si procesar estos tuits o por el contrario desecharlos. En este caso se ha decidido desechar los tuits para evitar atrasar la fecha que el usuario ha marcado. En este método se realizará también el cálculo de las estadísticas de los tuits que no hayan sido procesados con anterioridad.

```
this.eliminarTemporizadores();

twitterStream.removeListener(listener);
twitterStream.shutdown();
```

Figura 12: Código del temporizador 3

A continuación, vemos el método para eliminar los temporizadores.

```
private void eliminarTemporizadores(){
    Collection<Timer> timers = timer.getTimers();
    if (timers != null){
        Iterator iterator = timers.iterator();
        while(iterator.hasNext()){
            Timer t = (Timer) iterator.next();
            t.cancel();
        }
    }
}
```

Figura 13: Código del temporizador 4

La segunda tarea con temporizador será una tarea que se ejecutará de forma periódica y que calculará los datos que no hayan sido procesados hasta el momento. El uso de este temporizador se realizará de forma más sencilla que el anterior, puesto que sólo necesitaremos incluir una anotación en el método.

```
@Schedule(minute = "*/5", hour = "*", persistent = false)
```

Figura 14: Código del temporizador 5

La anotación *minute = "*/5"* indica que este método se ejecutará cada minuto múltiplo de 5, (en este ejemplo, en nuestro caso será de cada 20 minutos), por su parte la anotación *hour ="*"* implica que la restricción de los minutos impuesta anteriormente será aplicada a cara hora. Por ejemplo, en el ejemplo de la *Figura 14* se ejecutará el método en el minuto 5,10,15,20, …, de cada hora. Por último, la anotación *persistence=false* indica que cuando el servidor deja de estar activo no se salvan los temporizadores.

4.4 Cálculo de los datos

Para realizar el cálculo de las estadísticas se va a distinguir entre el procesamiento periódico y el último procesamiento. Esto se debe principalmente a dos motivos: para evitar interferencias con el *timer* y porque en el último procesamiento es cuando se realizará el análisis de la valoración (positiva o negativa), esto se explicará más adelante. En ambas ejecuciones la lógica será similar salvo la excepción comentada anteriormente.

El procesamiento se realizará tuit a tuit, por lo que necesitaremos obtenerlo de la base de datos (explicado en el punto 4.2). Una vez obtenido el texto del tuit y la fecha de creación vamos a separar ambos cálculos.

A través del texto del tuit vamos a extraer las tres cuentas más mencionadas y los tres hashtags más utilizados en relación al tema introducido por el usuario. Para almacenar estos datos se utilizará la estructura de datos *HashMap<String,Integer>* en el que el *String* representa el hashtag o la cuenta y el *Integer* el número de veces que ha aparecido en todos los tuits procesados. Se dividirá en *tokens* el texto del tuit (usando el espacio como delimitador) y, en caso de que el token comience por # o @, se sumará uno en la estructura de datos (o se añadirá en su defecto). Una vez que tenemos todos los datos introducidos en su correspondiente estructura vamos a realizar una ordenación (dicha función ha sido tomada y modificada de la página StackOverFlow, concretamente de la entrada: Entrada realizada el 19/6/2017) para tener los tres datos con mayor ocurrencia al principio. Nos basaremos en la estructura de datos *LinkedHashMap<String,Integer>,* la cual es una estructura muy similar a la anteriormente usada, pero mantiene el orden de inserción.

```
mongo.findAllDatos();
Map<String, Integer> hashtag = mongo.getHashtags();
Map<String, Integer> cuentas = mongo.getCuentas();

LinkedHashMap<String, Integer> hashtagOrdenado = calcularMapaOrdenado( hashtag);
LinkedHashMap<String, Integer> cuentasOrdenado = calcularMapaOrdenado( cuentas);
```

Figura 15: Código del cálculo de datos

Una vez que ya tenemos los datos ordenados obtenemos los 3 primeros de la lista y los guardamos en un *array* de tamaño 3 que luego utilizaremos para mostrarlos.

A través de la fecha vamos a extraer las horas en las que más se ha tuiteado sobre el tema. Para ello, también dividiremos en *tokens* la fecha para así obtener la hora en la que se creó el tuit y repetir el mismo proceso explicado anteriormente para las cuentas y los hashtags. En el caso de las horas, guardaremos las ocurrencias en un *array* de tamaño 23 que representa las horas del día (desde las 00 hasta las 23).

4.5 Cálculo de la valoración del tema

En un primer momento se pensó realizar esto mediante API.AI, pero debido a problemas con la complejidad del mensaje recibido y de la petición se ha decidido utilizar otro medio. El método elegido ha sido enviar una petición a un servicio REST llamado *text-processing sentiment analysis* que devuelve en un mensaje JSON si el texto enviado valora positiva, negativa o neutralmente cierto tema. En el caso de que el tuit sea neutral no se tomará en cuenta para la estadística.

La petición será POST y se realizará de la siguiente manera.

```
HttpURLConnection httpcon;
try {
    httpcon = (HttpURLConnection) ((new URL("http://text-processing.com/api/sentiment/").openConnection()));
    httpcon.setDoOutput(true);
    httpcon.setRequestProperty("Content-Type", "application/json");
    //httpcon.setRequestProperty("text", "good");
    httpcon.setRequestMethod("POST");
    httpcon.connect();
    DataOutputStream wr = new DataOutputStream(httpcon.getOutputStream());

    wr.writeBytes("text="+texto_tuit);
    wr.flush();
    wr.close();

    BufferedReader in = new BufferedReader(
    new InputStreamReader(httpcon.getInputStream()));
    String inputLine;
    StringBuffer response = new StringBuffer();

    while ((inputLine = in.readLine()) != null) {
        response.append(inputLine);
    }
    in.close();
```

Figura 16: petición de la valoración

Como aparece en la *Figura 16*, mediante un *BufferedReader* y un *InputStreamReader* podemos obtener la respuesta para su posterior procesamiento dependiendo si la respuesta es positiva o negativa.

4.6 Interfaz de gráfica de usuario

Para la parte de la interfaz hemos utilizado Java Servers Faces en combinación con Bootstrap. Se ha desarrollado un esquema general para que todas las vistas sigan la misma regla. Se han realizado pequeñas modificaciones en los CSSs para ajustar las posiciones de los elementos.

```
<ui:insert name="body"/>
<h:form>
<nav class="navbar navbar-inverse navbar-fixed-top">
  <div   class="container">
    <div id="navbar" class="collapse navbar-collapse">
      <ul class="nav navbar-nav">
        <li><h:commandLink value="Campaña" class="active" action="#{twitterBean.campana()}"/>  </li>
        <li><h:commandLink value="Datos Pasados" action="#{twitterBean.datosPasados()}"/></li>
        <li><h:commandLink value="Datos" action="#{twitterBean.verDatos()}"/></li>
        <li><a href="http://localhost:8080/twitterData-war/faces/Contacto.xhtml">Contacto</a></li>
      </ul>
    </div><!--/.nav-collapse -->
  </div>
</nav>
</h:form>
```

Figura 17: Código de la barra de navegación

Para la elaboración de gráficas se ha utilizado una librería de JavaScript que se explicará en el siguiente capítulo.

A continuación se muestran unas imágenes con las vistas diseñadas:

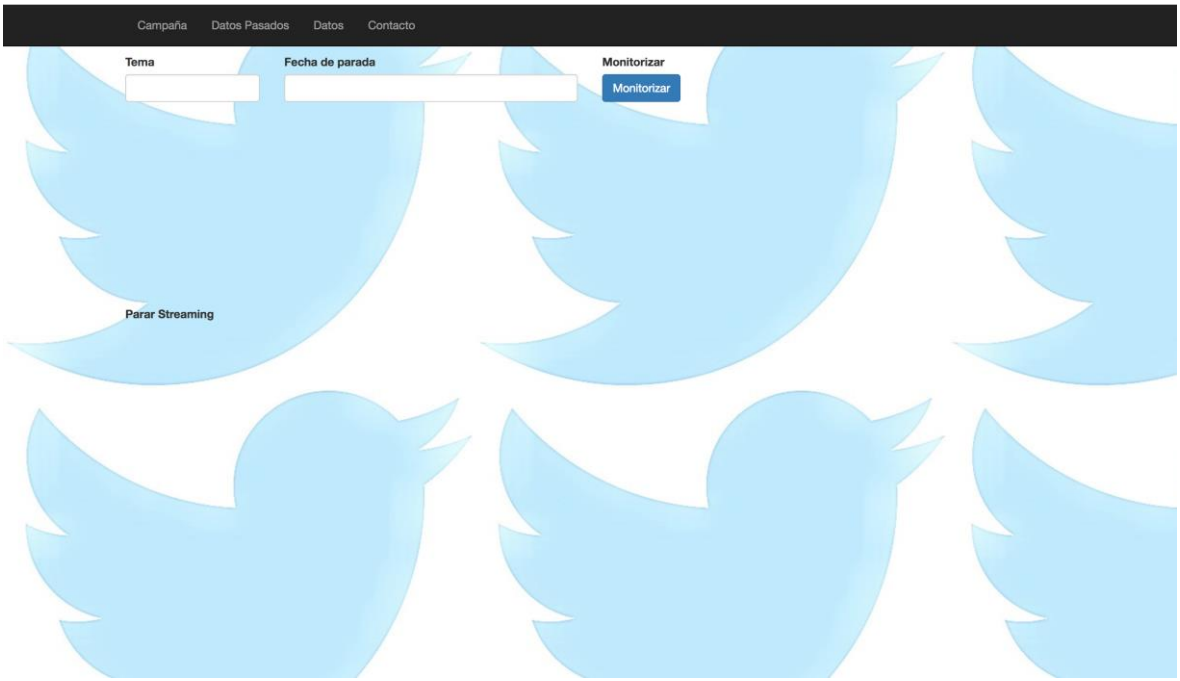

Figura 18: Vista Campaña sin iniciar

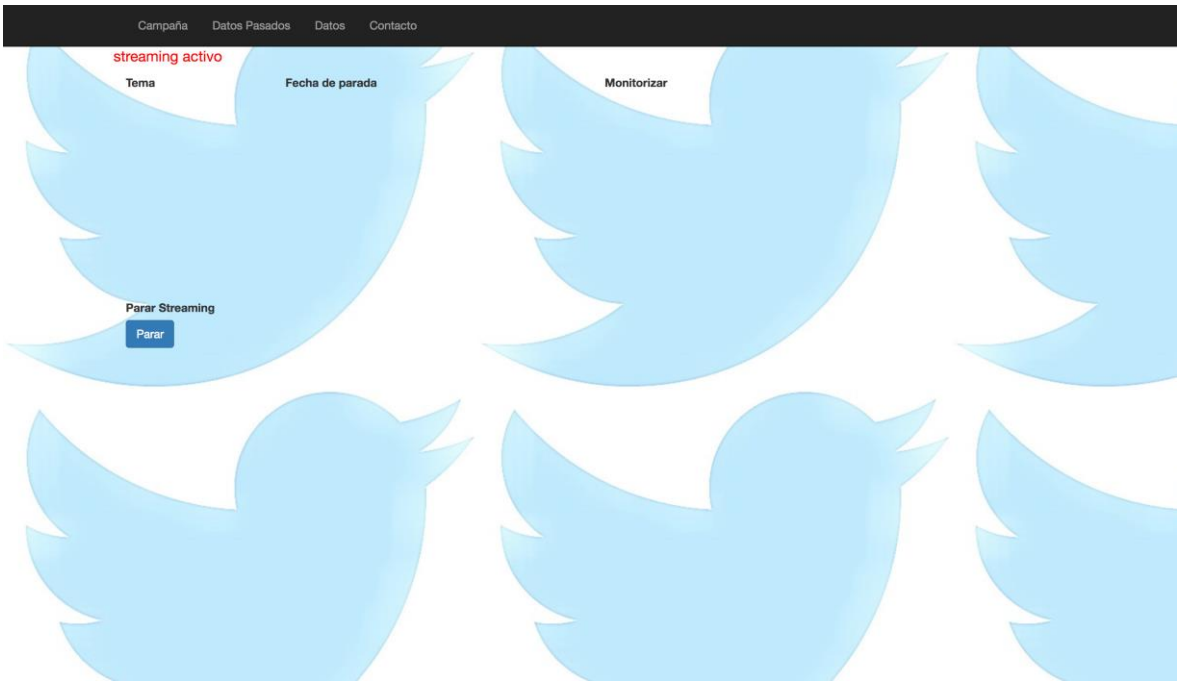

Figura 19: Vista Campaña iniciado

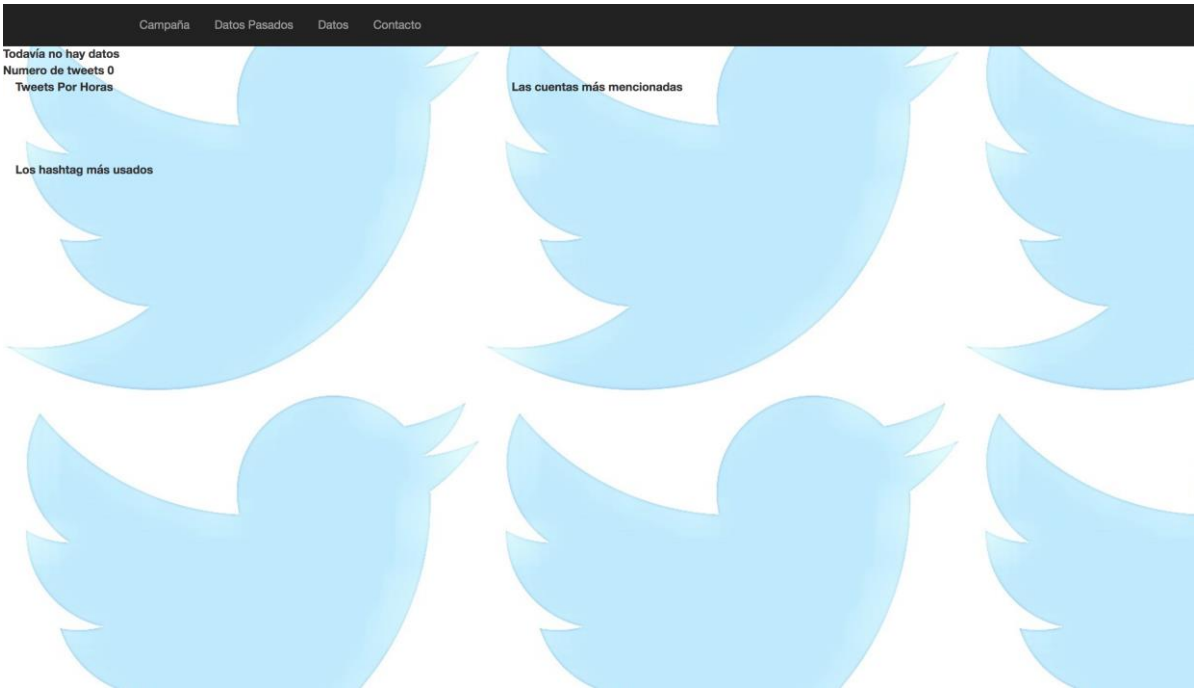

Figura 20: Vista Datos sin procesar

Figura 21: Vista Datos procesados

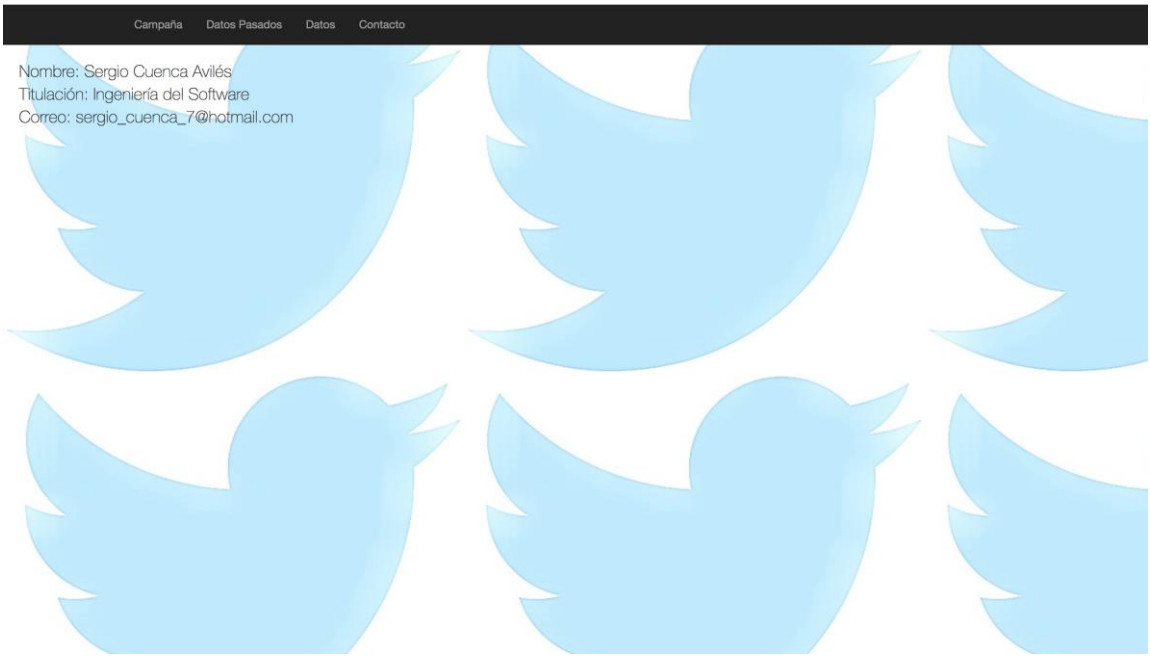

Figura 22: Vista Contacto

4.7 Dibujo de gráficas

Para el dibujo de las gráficas usaremos JavaScript y la librería C3.js la cual está basada en 3D.js pero siendo esta más sencilla e intuitiva de utilizar.

Utilizaremos dos tipos de gráficas: una en forma de “donut” y tres gráficas de barras verticales. Entre las gráficas de barras verticales distinguiremos entre dos tipos: una en la que se representa un dato a través de dos variables y otras dos en la que se presentan tres datos a través de una variable. Dichas gráficas tienen la posibilidad de ocultar cualquier variable si pulsamos encima del dato en la leyenda.

Para introducir las gráficas tenemos que crear un *div* en el *body* de la vista y ponerle un identificado, cuando se cargue la vista se introducirá en el *div* la gráfica que se ha creado mediante JavaScript. La librería exige que los datos se representen de la forma “[‘nombre Datos’, dato1, dato2,].

En la siguiente figura vemos un ejemplo de creación de una gráfica de barras verticales siendo chart el id del div.

```
var chart = c3.generate({
bindto: '#chart',
data: {
columns: [
//['data1', 30, 200, 100, 400, 150, 250]
cuentaTop1,
cuentaTop2,
cuentaTop3
],
type: 'bar'
},
bar: {
width: {
ratio: 0.6 // this makes bar width 50% of length between ticks
}
}
});
```

Figura 23: Código JavaScript gráficas

En este caso vamos a ver un ejemplo de creación de una gráfica en forma de "donut".

```
var chart4 = c3.generate({
    bindto: '#chart4',
data: {
    columns: [
        ['data1', #{twitterBean.negativo}],
        ['data2', #{twitterBean.positivo}]
    ],
    type : 'donut',
      colors: {
        data1: '#ff0000',
        data2: '#00ff00'
    }
},
donut: {
    title: "Valoracion"
}
});
```

Figura 24: Código JavaScript gráficas

Se va a proceder a la explicación de cada una de las gráficas.

La figura 25 muestra las tres cuentas más mencionadas en el tema determinado (en este caso Trump) durante el tiempo de monitorización. Si se pone el cursor

encima de la gráfica se expande un rectángulo dónde se especifica los valores exactos de cada dato y el color asociado.

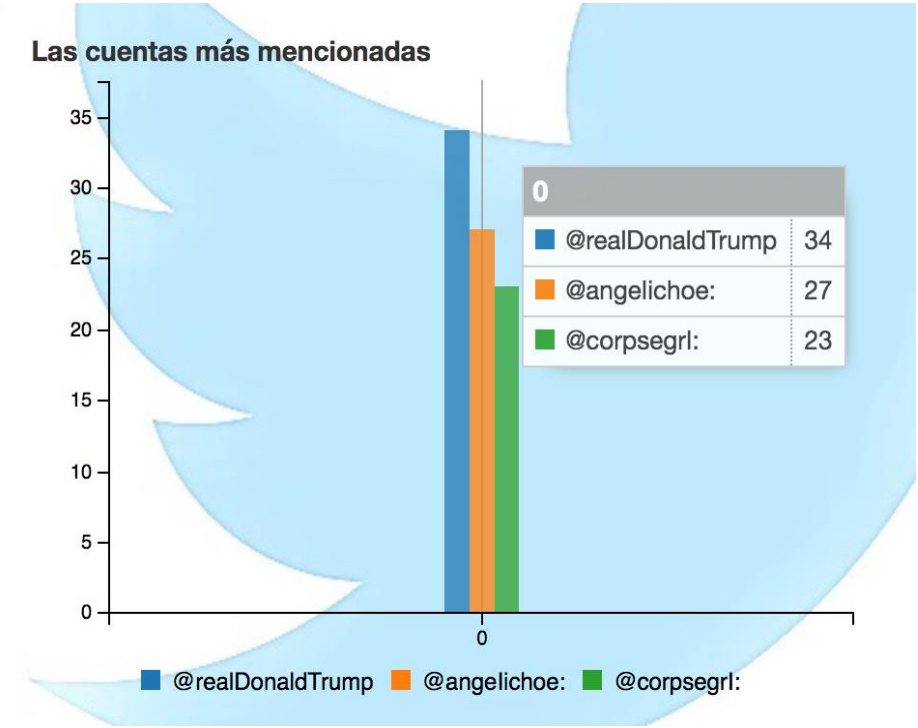

Figura 25: gráfica cuentas

Se ha seleccionado el color azul para el primero, el naranja para el segundo y el verde para el tercero.

La figura 26 muestra los 3 hashtags más utilizados en el tema determinado (en este caso Trump) durante el tiempo de monitorización. Si se posa el cursor encima de la gráfica se expande un rectángulo dónde se especifica los valores exactos de cada dato y el color asociado.

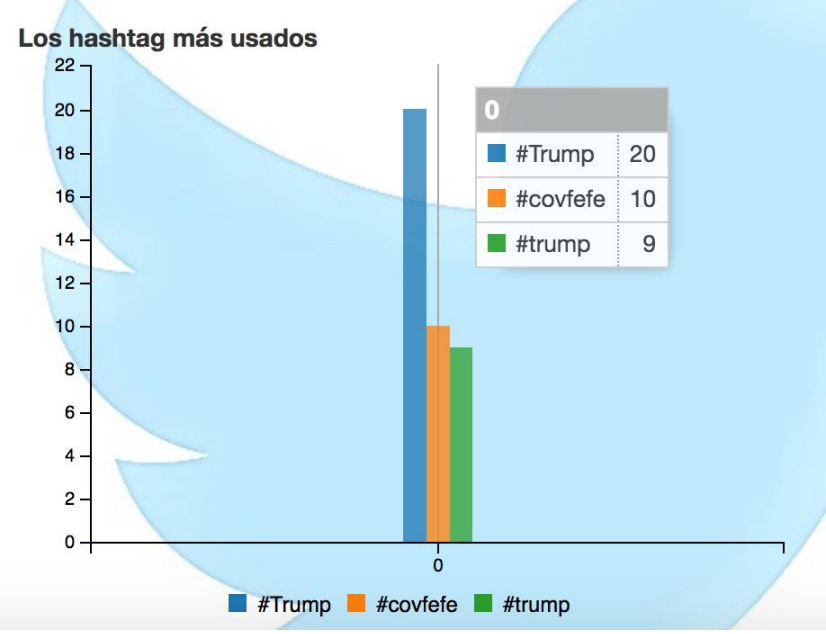

Figura 26: gráfica hashtags

La figura 27 muestra todas las horas del día y los tuits que se han recogido en esa hora durante el tiempo de ejecución, si la ejecución dura más de un día los tuis realizados a la misma hora se suman. En este caso también se expande un rectángulo con los valores exactos y los colores asociados. En este caso, como puede apreciarse, se ha utilizado un solo color.

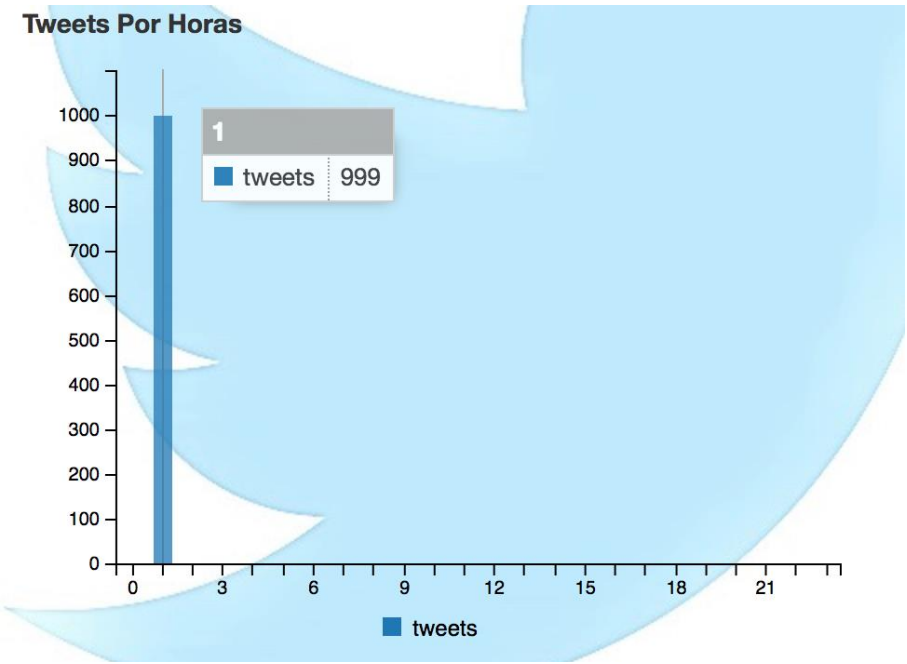

Figura 27: gráfica horas

La figura 28 muestra una gráfica en forma de "*donut*" en la que el color verde representa el porcentaje de valoraciones positivas y el color rojo el porcentaje de valoraciones negativas.

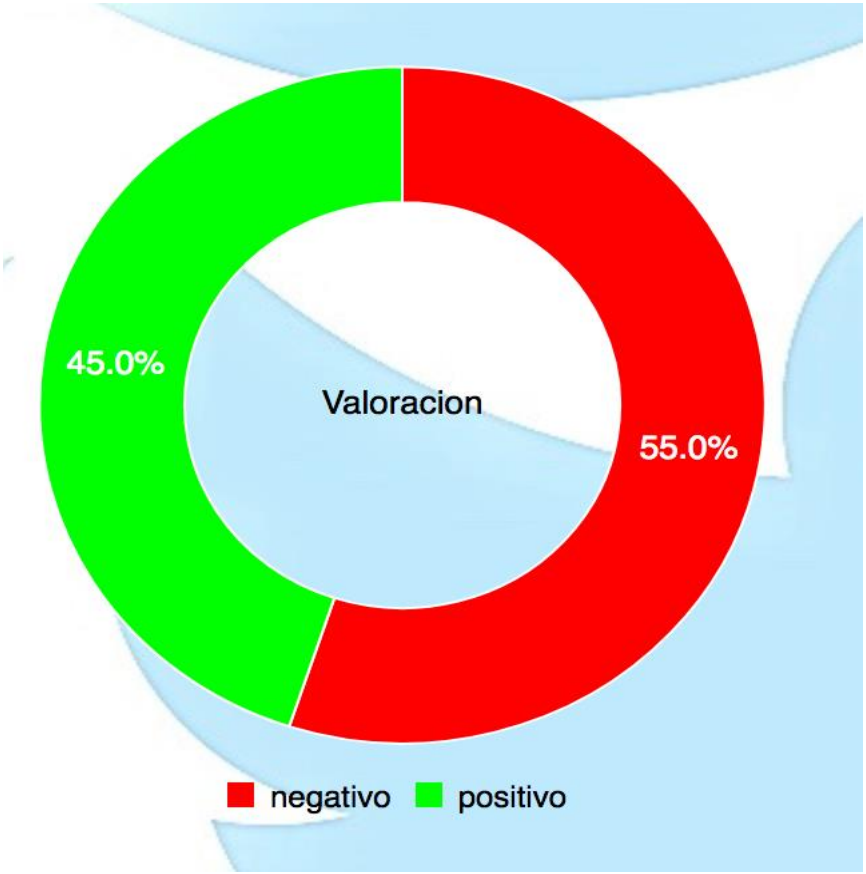

Figura 28: gráfica valoración

El porcentaje total suma 100%, cabe aclarar que realmente no son el 100% de los tuits, ya que como se ha comentado antes, se han suprimido de esta estadística aquellos tuits cuya valoración es neutral. Por tanto, podemos decir que el "donut" muestra un porcentaje sobre el 100% de la suma de los tuits positivos y negativos.

4.8 Extracción de tuits pasados

Otra funcionalidad que se ha implementado en nuestra aplicación, es la de obtener algunos datos estadísticos de tuis que han sido publicados en un momento anterior y que no podemos obtener a través de la monitorización. Recordemos que Twitter impone una restricción temporal por la que solo podemos obtener tuits con como máximo 7 días de atraso. Otra restricción a tener en cuenta, es que el número de tuits que podemos obtener en un periodo corto de tiempo también está muy limitado, por ese motivo se ha decidido analizar como máximo 1000 tuits para poder así obtener resultados de una manera rápida y evitar que Twitter prohíba el flujo de tuits durante un periodo de tiempo.

```
ConfigurationBuilder cb = new ConfigurationBuilder();
cb.setDebugEnabled(true)
        .setOAuthConsumerKey("XXXXXX")
        .setOAuthConsumerSecret("XXXX")
        .setOAuthAccessToken("XXXXXXX")
        .setOAuthAccessTokenSecret("XXXXXX")
        .setHttpProxyPort(3128)
        .setHttpProxyHost("proxy.wifi.uma.es")
        .setHttpProxyUser("proxy.wifi.uma.es");
twitter = new TwitterFactory(cb.build()).getInstance();
```

Figura 29: código tuits pasado

Este proceso de ejecución es totalmente distinto a todo el explicado anteriormente, por tanto, puede ejecutarse a la vez que la monitorización, no utiliza ninguna base de datos.

La autenticación se realiza utilizando las mismas credenciales que en el *Streaming*, pero en esta ocasión creamos un objeto del tipo *Twitter.*

Para obtener los tuits se utiliza una *query* en la que se introduce el tema y con un cursor para iterar a través de los tuits e ir almacenando los datos en una lista para su posterior procesamiento.

```
Query query =  new Query(tema);
QueryResult resultado;
```

Figura 30: código tuit pasado 2

```
    do {
        restriccion = listaTweets.size();
        resultado = twitter.search(query);
        listaTweets.addAll(resultado.getTweets());
        System.out.println("PASO ");
    } while (restriccion < limite && (query = resultado.nextQuery()) != null);
```

Figura 31: código tuit pasado 3

El procesamiento y cálculo de las estadísticas se realizan de la misma manera que para el *Streaming*, en esta ocasión calcularemos las cuentas más mencionadas y los hashtags más utilizados.

Del mismo modo, la interfaz gráfica ha sido diseñada de la misma manera que para el ejemplo anterior.

En la siguiente figura vemos un ejemplo de ejecución utilizando la marca Adidas.

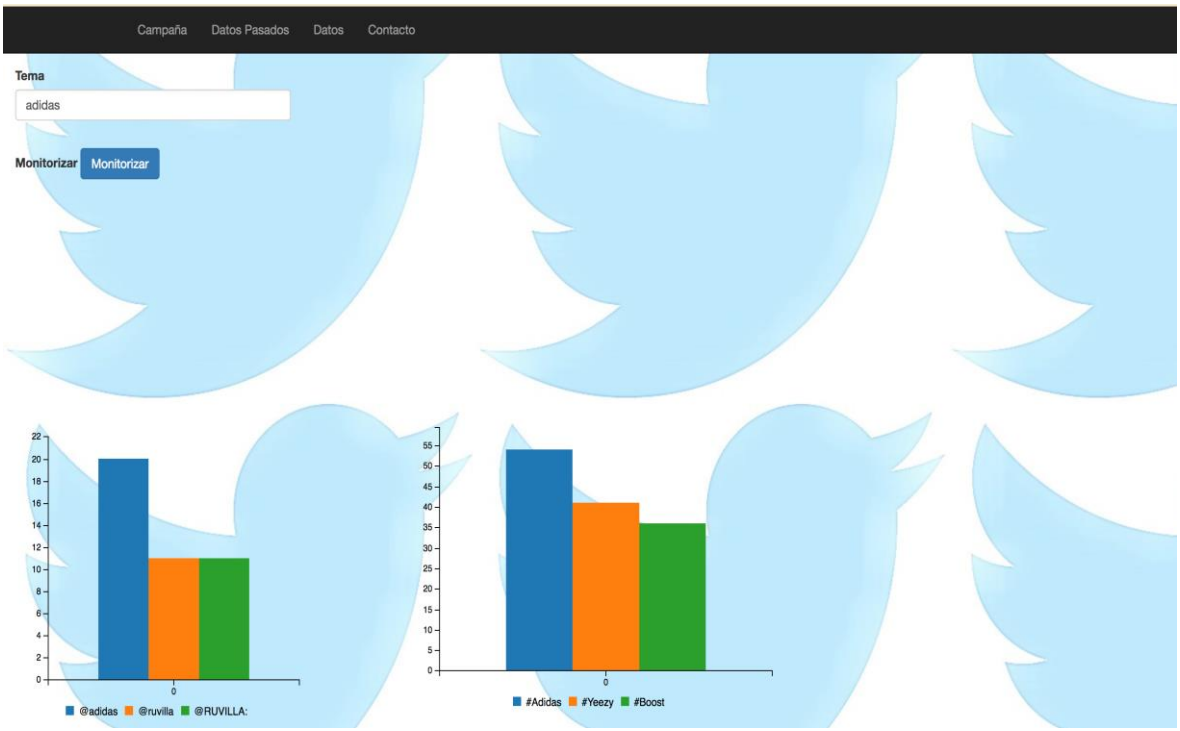

Figura 32: gráficas tuits pasados

Se ha decidido eliminar la estadística de las horas ya que solo se van a procesar 1000 tuits y esto hace que dicha estadística carezca de valor. En el caso de la valoración positiva o negativa, también se ha eliminado debido a que cada petición REST consume un tiempo relativamente elevado y alargaría más la espera para la obtención de estos resultados además de evitar sobrepasar el número máximo permitido de peticiones diarias.

5. Pruebas realizadas

Se han realizado una gran cantidad de pruebas utilizando temas en los que la afluencia de tuits es casi ilimitada (e.g Trump) para así someter a nuestro sistema a un estrés alto y conseguir realizar la aplicación con el menor número de *bugs*.

Como resultado de estas pruebas se han obtenido algunos bugs que posteriormente han sido solucionados. A continuación se expone una lista de bugs que se han encontrado tras la realización de estas pruebas y su pertinente solución

- Cuando el *Streaming* estaba activo durante un periodo temporal amplio la sesión del navegador expiraba produciendo así un error. La solución propuesta ha sido cambiar el ámbito del *Bean* para que la sesión se mantenga viva durante toda la ejecución del programa.
- Cuando el *Streaming* estaba activo durante un periodo temporal amplio y además la afluencia de tuits era enorme se producía un error de memoria. La solución se ha realizado en dos pasos. En primer lugar, se ha ampliado la memoria del entorno de desarrollo, así mismo, se ha establecido un límite máximo de tuits (100.000 tuits) a monitorizar debido a que la memoria de nuestra máquina no es ilimitada.
- En un primer momento las peticiones al servicio de valoración se realizaban durante toda la ejecución. cuando las peticiones se alargaban en el tiempo por motivos de conexión o de límites se producías desajustes en el procesado de datos. La solución a esto ha sido la de calcular este dato al final de la monitorización.

6. Conclusiones y Líneas Futuras

En este apartado se van a tratar las conclusiones obtenidas tanto personales como académicas dividas en distintas áreas.

- Obtención de requisitos: Una de las fases más importantes a la hora de realizar un proyecto, en el caso de no tener claro alguno de los requisitos necesarios podría causar la realización de una gran cantidad de cambios perdiendo así mucho tiempo.

- Diseño de las interfaces: Una de las partes más difíciles de todo el ámbito del desarrollo a mi parecer, pues es algo que tiene que gustar a los usuarios de la aplicación. Personalmente es la parte que menos me gusta de mi proyecto.

- Implementación: La parte más satisfactoria de todas, aunque también la más laboriosa. Cuando se realiza un proyecto de estas dimensiones, de forma autónoma, entiendes realmente la importancia de la reutilización del código y no centrarte sólo en las partes “nuevas” y diferenciales de tu proyecto, dándole más importancia a librerías ya creadas.

- Documentación: Esta parte es la menos atractiva de todo el proyecto, pero no por ello deja de ser importante. En un desarrollo de estas dimensiones temporales consigues ver la importancia de tener una documentación realizada a la misma vez que el código.

- Pruebas: Igual que en las áreas anteriores, hasta que no tienes un proyecto de gran tamaño no consigues entender la verdadera importancia de realizar pruebas para obtener un producto con el menor número de errores y con el mejor funcionamiento posible.

Aparte de estos puntos, personalmente he aprendido muchas tecnologías que no había usado con anterioridad (lo cual ha producido un gasto mayor en horas de aprendizaje de las nuevas tecnologías de lo esperado) como, por ejemplo,

MongoDB, Bootstrap, librerías como Twitter4j o C3.js, etc. También me ha aportado una capacidad de organización mayor de la que creía poseer a la hora de trabajar.

En general, la experiencia obtenida con la realización de este proyecto ha sido bastante satisfactoria y productiva.

Una vez llegado a este punto podemos decir que nuestra aplicación está finalizada, pero existen muchos aspectos que podrían dar lugar a nuevas líneas de desarrollo con el objetivo de aumentar las funcionalidades.

En primer lugar, una de las líneas futuras que más mejoraría nuestra aplicación sería implementar un sistema de valoración de los tuits que se ejecute de forma local y no depender así de límites de peticiones ni tener en cuenta el tiempo de respuesta de peticiones REST.

La interfaz es otro punto que en caso de seguir desarrollando nuestra aplicación se podría mejorar haciéndola más elaborada.

Otra idea interesante que podría llevarse a cabo sería implementar un sistema de usuarios por la que cada uno pudiera tener un *streaming* distinto activo y poder consultar los datos en cualquier momento.

Bibliografía

http://twitter4j.org/javadoc/twitter4j/ - Twitter4j Javadoc [Accedido el 19/7/17]

https://github.com/yusuke/twitter4j/tree/master/twitter4j-examples - Twitter4j examples -GitHub [Accedido el 19/7/17]

http://theopentutorials.com/tutorials/java-ee/ejb3/ejb3-timer-service/ - one – EJB Timer Service – The Open Tutorials [Accedido el 19/7/17]

https://es.wikipedia.org/wiki/JavaServer_Faces JavaServer Faces - Wikipedia, la enciclopedia libre - [Accedido el 19/7/17]

http://c3js.org/examples.html Examples - C3js - [Accedido el 19/7/17]

https://docs.mongodb.com/v3.2/ Docs MongoDB - [Accedido el 19/7/17]

https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-os-x/ Install MongoDB on OS X – Docs MongoDB [Accedido el 19/7/17]

https://www.w3schools.com/bootstrap/default.asp Boostrap 3 - W3Schools [Accedido el 19/7/17]

https://stackoverflow.com/questions/34962677/twitter-streaming-api-limits Twitter Streaming Api Limits - StackOverFlow [Accedido el 19/7/17]

https://dev.twitter.com/docs Twitter Development Documentation - Dev Twitter - [Accedido el 19/7/17]

https://stackoverflow.com/questions/8119366/sorting-hashmap-by-values Sorting HashMap by value - StackOverFlow [Accedido el 19/7/17]

https://www.adictosaltrabajo.com/tutoriales/ejb-3-timer-service/ EJB 3 Timer Service – Adictos al trabajo [Accedido el 19/7/17]

http://text-processing.com/docs/sentiment.html Sentiment Analysis - Text Processing [Accedido el 19/7/17]

https://www.codementor.io/howardungar/getting-started-with-bootstrap-and-jsf-6bofntdov Getting started with boostrap and jsf – Codementor [Accedido el 19/7/17]